QUEENSLAND UNIVERSITY OF TECHNOLOGY

IFN 645 – Data Mining Technology and Applications

Assessment Item 1

Project Name: The analysis of potential organic product buyers

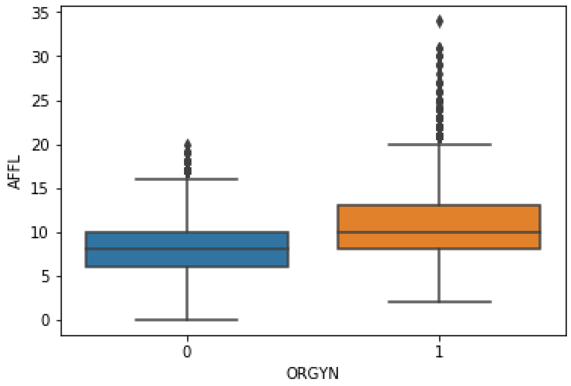
Team name: Data mining novices

Yen-Chang, Chen (n9861718)
Yixi Zhou (n9599291)
Anthony Lam (n9778977)

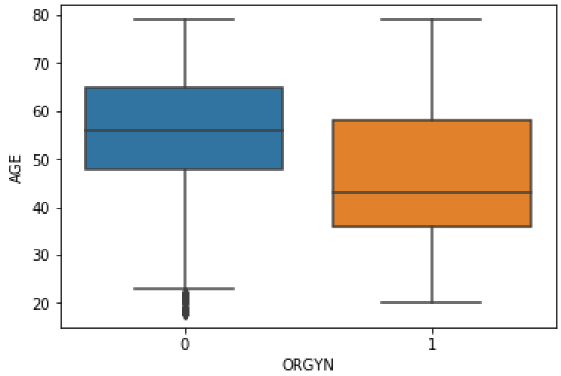
Task 1. Data Selection and Distribution

1. **Can you identify any clear patterns by initial exploration of the data using histogram or box plot?**

A: The following box plot shows the relationship between ORGYN and AFFL. It appears that organic product byers have relatively higher affluence grade than customers who do not purchase the organic products.



Additionally, the following box plot shows the relationship between ORGYN and AGE. It appears that most organic product buyers are in their age 35 to 60.



2. **What is the proportion of individuals who purchased organic products?**

A: The number of people who did not purchase the organic products is 16718 and the number of people who bought the products is 5505. Therefore, the proportion can be calculated through 5505 / (16718 + 5505), which approximately equals to ¼ of total participants.

```
df["ORGYN"].value_counts()
```

```
0    16718
1     5505
Name: ORGYN, dtype: int64
```

3. **Did you have to fix any data quality problems? Detail them.**

A: Some values are missing, including GENDER, AGE, AGEGRP1, AGEGRP2, TV_REG, NEIGHBORHOOD, LCDATE, NGROUP, REGION, AFFL and LTIME. Some of them might be useful for the later analysis. Therefore, some rectifications need to be taken to address this issue. In fact, different techniques are employed in processing missing values for different attributes. For GENDER, it has a considerable amount of missing values. Thus, dropping this column would be a better option. In terms of AGE, this column has some missing value, but those missing value could be calculated using EDATE to minus DOB. With this measure, the AGE column could be collected without any missing value. The rest of the attributes have only a small amount of missing values, so it is acceptable to drop a few rows. This approach would not impact the model construction significantly.

Incorrect data type is also found in our dataset. CLASS should be considered as an ordinal value since it indicates hierarchical ranking system for the loyalty status of each customer. In CLASS column, “Tin”, “Silver”,” Gold” and “Platinum” are replaced by zero, one, two and three respectively. In addition, to CLASS, NEIGHBORHOOD is using incorrect data type. The values within the NEIGHBORHOOD column should be regarded as nominal values as they indicate different regions using numerical value. There is no relationship between those values.

Some erroneous values are in BILL and AFFL. A large number of values which are smaller than one in BILL are unusual in the data. In terms of AFFL, the scale of affluent grade should be from 1~30 but the following image shows some values that either are over 30 or less than one. In BILL case, the values which are smaller than one are removed and replace them with the mean of BILL. For AFFL, the missing values and numbers that are over 30 or less than one are replaced by its mean as well.

```
########   DATA PREPROCESSING   ########

import numpy as np
import pandas as pd

def data_prep():

    #READ ORGANICS FILE
    df = pd.read_csv('datasets/organics.csv')

    #CHANGE CLASS INTO ORDINAL VALUE
    class_map = {"Tin":0, "Silver":1, "Gold":2, "Paltinum":3}
    df["CLASS"] = df["CLASS"].map(class_map)

    #AGE IMPUTATION
    df["DOB"] = pd.to_datetime(df["DOB"])
    df["EDATE"] = pd.to_datetime(df["EDATE"])
    df["AGE"] = df["EDATE"] - df["DOB"]
    df["AGE"] = pd.to_timedelta(df["AGE"])
    df["AGE"] = (df["AGE"] / np.timedelta64(1, 'D')).astype(int)//365

    #BILL VALUE CORRECTION
    mask = df["BILL"] < 1
    df.loc[mask, "BILL"] = np.nan
    df["BILL"].fillna(df["BILL"].mean(), inplace = True)

    #AFFL VALUE CORRECTION
    mask1 = (df["AFFL"] > 30) | ((df["AFFL"] < 1))
    df.loc[mask1, "AFFL"] = np.nan
    df["AFFL"].fillna(df["AFFL"].mean(), inplace = True)

    #CONVERTING NEIGHBORHOOD TO STRING
    df["NEIGHBORHOOD"] = df["NEIGHBORHOOD"].astype(str)

    #DROP UNUSED ATTRIBUTE AND UNUSED TARGET VARIABLE
    df.drop(["CUSTID", "GENDER", "DOB", "EDATE", "AGEGRP1", "AGEGRP2", "LCDATE", "ORGANICS", "NGROUP"], axis = 1, inplace =
True)

    #DROP THE ROWS WITH MISSING VALUE
    df = df.dropna(axis = 0, how = 'any')

    #ONE HOT ENCODING
    df = pd.get_dummies(df)

    return df
```

4. **What variables did you include in the analysis and what were their roles and measurement level set? Justify your choice.**

A: Several variables are selected for the analysis which are AGE, BILL CLASS, ORGYN, AFFL, LTIME, TV_REG, NEIGHBORHOOD, REGION. The others are excluded because some of them are useless for later data analysis (CUSTID and EDATE) or have similar attribute with another variable. For example, AGE, DOB, AGEGRP1 and AGEGRP2 provide similar information but with disparate representation. Therefore, only AGE is selected. The same reason applies to “LCDATE, LTIME” and “NGROUP, NEIGHBORHOOD”. In terms of target variable, the ORGYN is used because the aim of this case study is to identify the potential organic product buyers within over 20,000 participants. Thus, the target variable should belong to binary classification, which is ORGYN. Hence, the ORGANICS is dropped accordingly. The other selected variables are expected to produce insightful result based on their individual unique attribute. Those attribute aids manager to understand its target customer.

The following images show the measurement level set of each input variable.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20662 entries, 0 to 22222
Data columns (total 80 columns):
AGE                  20662 non-null int32
BILL                 20662 non-null float64
CLASS                20662 non-null float64
ORGYN                20662 non-null int64
AFFL                 20662 non-null float64
LTIME                20662 non-null float64
TV_REG_Border        20662 non-null uint8
TV_REG_C Scotland    20662 non-null uint8
TV_REG_East          20662 non-null uint8
TV_REG_London        20662 non-null uint8
TV_REG_Midlands      20662 non-null uint8
TV_REG_N East        20662 non-null uint8
TV_REG_N Scot        20662 non-null uint8
TV_REG_N West        20662 non-null uint8
TV_REG_S & S East    20662 non-null uint8
TV_REG_S West        20662 non-null uint8
TV_REG_Ulster        20662 non-null uint8
TV_REG_Wales & West  20662 non-null uint8
TV_REG_Yorkshire     20662 non-null uint8
NEIGHBORHOOD_1.0     20662 non-null uint8
NEIGHBORHOOD_10.0    20662 non-null uint8
NEIGHBORHOOD_11.0    20662 non-null uint8
NEIGHBORHOOD_12.0    20662 non-null uint8
NEIGHBORHOOD_13.0    20662 non-null uint8
NEIGHBORHOOD_14.0    20662 non-null uint8
NEIGHBORHOOD_15.0    20662 non-null uint8
NEIGHBORHOOD_16.0    20662 non-null uint8
NEIGHBORHOOD_17.0    20662 non-null uint8
NEIGHBORHOOD_18.0    20662 non-null uint8
NEIGHBORHOOD_19.0    20662 non-null uint8
NEIGHBORHOOD_2.0     20662 non-null uint8
NEIGHBORHOOD_20.0    20662 non-null uint8
NEIGHBORHOOD_21.0    20662 non-null uint8
NEIGHBORHOOD_22.0    20662 non-null uint8
NEIGHBORHOOD_23.0    20662 non-null uint8
NEIGHBORHOOD_24.0    20662 non-null uint8
NEIGHBORHOOD_25.0    20662 non-null uint8
NEIGHBORHOOD_26.0    20662 non-null uint8
NEIGHBORHOOD_27.0    20662 non-null uint8
NEIGHBORHOOD_28.0    20662 non-null uint8
NEIGHBORHOOD_29.0    20662 non-null uint8
NEIGHBORHOOD_3.0     20662 non-null uint8
NEIGHBORHOOD_30.0    20662 non-null uint8
NEIGHBORHOOD_31.0    20662 non-null uint8
NEIGHBORHOOD_32.0    20662 non-null uint8
NEIGHBORHOOD_33.0    20662 non-null uint8
NEIGHBORHOOD_34.0    20662 non-null uint8
NEIGHBORHOOD_35.0    20662 non-null uint8
NEIGHBORHOOD_36.0    20662 non-null uint8
NEIGHBORHOOD_37.0    20662 non-null uint8
NEIGHBORHOOD_38.0    20662 non-null uint8
NEIGHBORHOOD_39.0    20662 non-null uint8
NEIGHBORHOOD_4.0     20662 non-null uint8
NEIGHBORHOOD_40.0    20662 non-null uint8
NEIGHBORHOOD_41.0    20662 non-null uint8
NEIGHBORHOOD_42.0    20662 non-null uint8
NEIGHBORHOOD_43.0    20662 non-null uint8
NEIGHBORHOOD_44.0    20662 non-null uint8
NEIGHBORHOOD_45.0    20662 non-null uint8
NEIGHBORHOOD_46.0    20662 non-null uint8
NEIGHBORHOOD_47.0    20662 non-null uint8
NEIGHBORHOOD_48.0    20662 non-null uint8
NEIGHBORHOOD_49.0    20662 non-null uint8
NEIGHBORHOOD_5.0     20662 non-null uint8
NEIGHBORHOOD_50.0    20662 non-null uint8
NEIGHBORHOOD_51.0    20662 non-null uint8
NEIGHBORHOOD_52.0    20662 non-null uint8
NEIGHBORHOOD_53.0    20662 non-null uint8
NEIGHBORHOOD_54.0    20662 non-null uint8
NEIGHBORHOOD_55.0    20662 non-null uint8
NEIGHBORHOOD_6.0     20662 non-null uint8
NEIGHBORHOOD_7.0     20662 non-null uint8
NEIGHBORHOOD_8.0     20662 non-null uint8
NEIGHBORHOOD_9.0     20662 non-null uint8
NEIGHBORHOOD_nan     20662 non-null uint8
REGION_Midlands      20662 non-null uint8
REGION_North         20662 non-null uint8
REGION_Scottish      20662 non-null uint8
REGION_South East    20662 non-null uint8
REGION_South West    20662 non-null uint8
dtypes: float64(4), int32(1), int64(1), uint8(74)
```

5. **What distribution scheme did you use? What data partitioning allocation did you set? Explain your selection.**

A: For distribution scheme, the stratification is utilized to distribute same proportion of positive and negative target value (ORGYN) to two different datasets (test and train). This measure provides more reliable result for data analysis. In this case study, training data is set for 70% and testing data is set for 30%. Those ratios are identical to the exercise we did in the practical session because it is memorable and also common in most of the data splitting allocation technique.

```
#TARGET/IMPUT SPLIT
y = df["ORGYN"]
X = df.drop(["ORGYN"], axis = 1)

#SETTING RANDOM STATE
rs = 10

#DATA PARTITION
X_mat = X.as_matrix()
X_train, X_test, y_train, y_test = train_test_split(X_mat, y, test_size = 0.3, stratify = y, random_state = rs)
```

Task 2. Predictive Modelling Using Decision Trees

1. **Build a decision tree using the default setting. Examine the tree results and answer the followings:**

   a. **What is the classification accuracy on training and test datasets?**

      The train accuracy for default decision tree is 0.9994468644126392

      The test accuracy for default decision tree is 0.7144700758186804

```
#SIMPLE DECISION TREE TRAINING
model = DecisionTreeClassifier(random_state = rs)
model.fit(X_train, y_train)

print("Train accuracy", model.score(X_train, y_train))
print("Test accuracy", model.score(X_test, y_test))
```

```
Train accuracy 0.9994468644126392
Test accuracy 0.7144700758186804
```

b. **What is the size of tree (i.e. number of nodes)?**

The number of nodes in default decision tree is 6437

c. **How many leaves are in the tree that is selected based on the validation data set?**

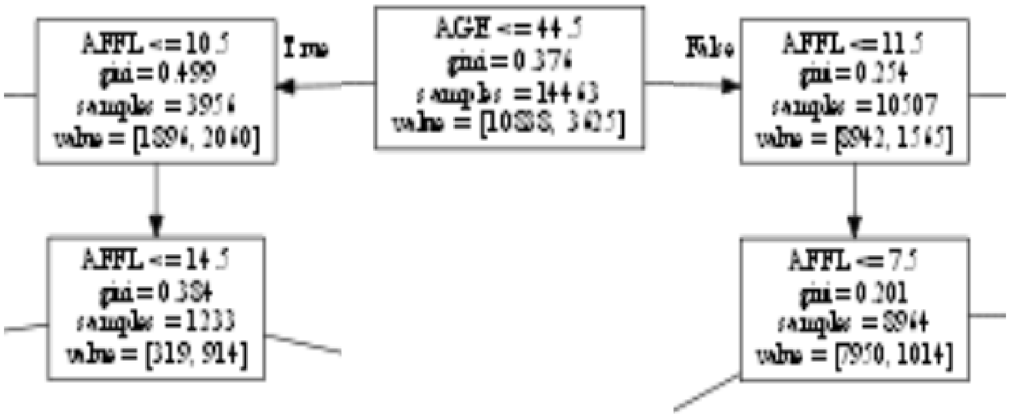
The leaf count is 3219

```
from data_preprocessing import get_tree_stats
get_tree_stats(model)
```

```
depth: 58
nodes: 6437
leaf count: 3219
```

d. **Which variable is used for the first split? What are the competing splits for this first split?**

Age < = 44.5 is the variable for the first split. The competing splits for this first split are AFFL <= 10.5 and AFFL <= 11.5



e. **What are the 5 important variables in building the tree?**

The most important variables in default tree are AGE, AFFL, BILL, LTIME, and CLASS.

```
import numpy as np
importances = model.feature_importances_
feature_names = X.columns

indices = np.argsort(importances)
indices = np.flip(indices, axis = 0)

indices = indices[:5]

for i in indices:
    print(feature_names[i], ":", importances[i])
```

```
AGE : 0.281620968745
AFFL : 0.138334709849
BILL : 0.100116381209
LTIME : 0.0840694126167
CLASS : 0.025128167679
```

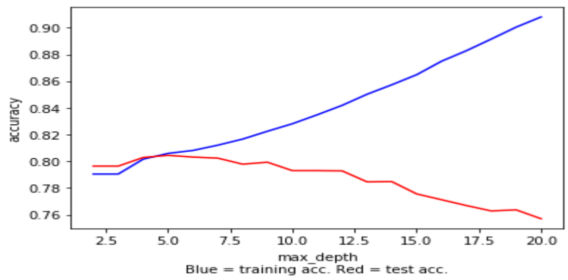
f. **Report if you see any evidence of model overfitting.**

The sign of overfitting can be discover through classification accuracy. The train accuracy is nearly 30% higher than the test accuracy, which means our model fits train data too well. Therefore, this model cannot produce trustworthy prediction.

g. **Did changing the default setting (i.e., only focus on changing the setting of the number of splits to create a node) help improving the model? Answer the above questions on the best performing tree.**

After we check the model performance for max depth from 2 to 20, the following image illustrates the max depth of best performing tree is approximately five.

```
import matplotlib.pyplot as plt

test_score = []
train_score = []

# check the model performance for max depth from 2-20
for max_depth in range(2, 21):
    model = DecisionTreeClassifier(max_depth=max_depth, random_state=rs)
    model.fit(X_train, y_train)

    test_score.append(model.score(X_test, y_test))
    train_score.append(model.score(X_train, y_train))

# plot max depth hyperparameter values vs training and test accuracy score
plt.plot(range(2, 21), train_score, 'b', range(2,21), test_score, 'r')
plt.xlabel('max_depth\nBlue = training acc. Red = test acc.')
plt.ylabel('accuracy')
plt.show()
```



By changing max depth to five, the results are shown as below. (Best performance tree)

```
#MAX_DEPTH = 5

model = DecisionTreeClassifier(max_depth = 5, random_state = rs)
model.fit(X_train, y_train)

print("Train accuracy:", model.score(X_train, y_train))
print("Test accuracy:", model.score(X_test, y_test))

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

importances = model.feature_importances_
feature_names = X.columns

indices = np.argsort(importances)
indices = np.flip(indices, axis = 0)

indices = indices[:5]

for i in indices:
    print(feature_names[i], ":", importances[i])
```

```
Train accuracy: 0.805918550785
Test accuracy: 0.804484594289
             precision    recall  f1-score   support

          0       0.82      0.94      0.88      4645
          1       0.69      0.40      0.51      1554

avg / total       0.79      0.80      0.79      6199

AGE : 0.606102275134
AFFL : 0.369933538826
BILL : 0.0074453348669
TV_REG_N West : 0.00569330552879
NEIGHBORHOOD_12.0 : 0.00415544944438
```

```
from data_preprocessing import get_tree_stats
get_tree_stats(model)
```

```
depth: 5
nodes: 63
leaf count: 32
```

The results shows that the train and test accuracy are 0.805918550785 and 0.804484594289 respectively. This information indicates no sign of overfitting in our model. Additionally, the test accuracy is slightly higher than previous maximal tree. Thus, it is believed that changing the number of split can be beneficial to model improvement.

2. **Build another decision tree tuned with GridSearchCV. Examine the tree results.**

```
#GridSearchCV1

from sklearn.model_selection import GridSearchCV

params = {'criterion': ["gini", "entropy"],
          "max_depth": range(2,7),
          "min_samples_leaf": range(10, 500, 10)}
cv = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=rs), cv=10)
cv.fit(X_train, y_train)

print("Train accuracy:", cv.score(X_train, y_train))
print("Test accuracy:", cv.score(X_test, y_test))

# test the best model
y_pred = cv.predict(X_test)
print(classification_report(y_test, y_pred))

# see importance
importances = model.feature_importances_
feature_names = X.columns

indices = np.argsort(importances)
indices = np.flip(indices, axis = 0)

indices = indices[:5]

for i in indices:
    print(feature_names[i], ":", importances[i])

# print parameters of the best model
print(cv.best_params_)
```

```
Train accuracy: 0.805019705455
Test accuracy: 0.803516696241
             precision    recall  f1-score   support

          0       0.83      0.93      0.88      4645
          1       0.67      0.43      0.52      1554

avg / total       0.79      0.80      0.79      6199

AGE : 0.626429834258
AFFL : 0.373090836468
BILL : 0.000479329274181
NEIGHBORHOOD_2.0 : 0.0
NEIGHBORHOOD_13.0 : 0.0
{'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 140}
```

a. **What is classification accuracy on training and test datasets?**

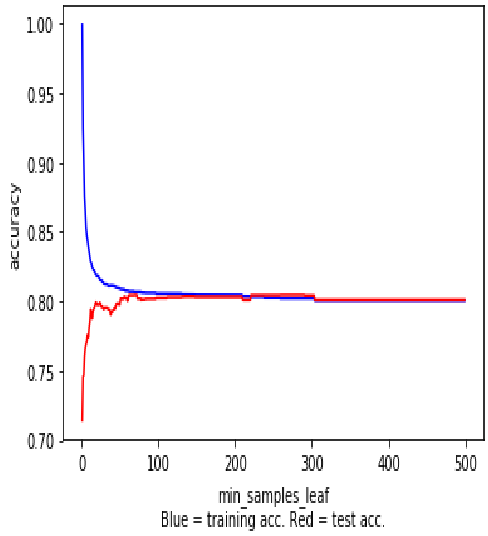
The train accuracy is 0.805227131301

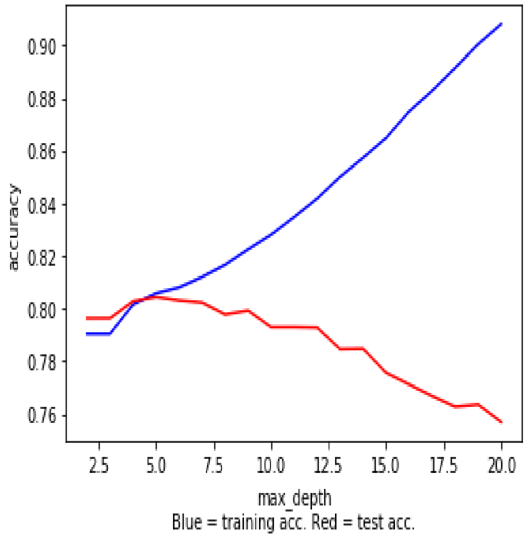
The Test accuracy is 0.805291175996

b. **What are the parameters used? Explain your decision.**

A: In this GridSearchCV, we used criterion, max_depth and min_samples_leaf to tune the model. In terms of criterion, we selected entropy and gini because they are commonly used in quality evaluation of a split. Setting max_depth can prevent the model from overfitting. Min_samples _leaf allows us to control the number of split times.

The following graphs shows the reason of selecting specific range in max_depth and min_samples_leaf (max_depth: 2~7 and min_samples_leaf: 10~500 with step of 10). In those particular range, the model shows better performance.





c. **What are the optimal parameters for this decision tree?**

A: The optimal parameters:

Criterion: gini, max_depth: 6, min_samples_leaf: 140

d. **What is the size of tree (i.e. number of nodes)? Is the size different from the tree built in the previous step (2.1)? Why?**

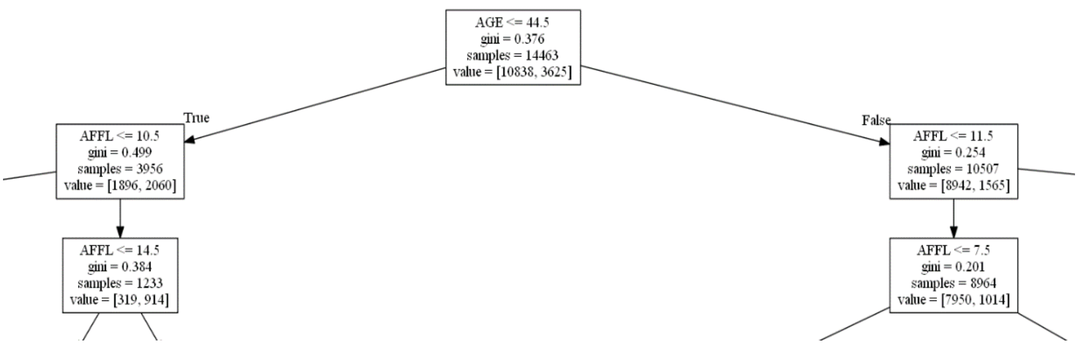
A: The number of nodes in this CV tree is 83. The size of CV tree has major difference comparing to default tree because the default tree has no growing restriction. In CV tree, we limits its max_depth and min_samples_leaf, which constrains its growth.

```python
from data_preprocessing import get_tree_stats
get_tree_stats(cv.best_estimator_)
```

```
depth: 6
nodes: 83
leaf count: 42
```

e. **Which variable is used for the first split? What are the competing splits for this first split?**

A: The answer of this question is identical to 2.1.e. The variable for the first split Is AGE <= 44.5. The competing splits for the first split are AFFL <= 10.5 and AFFL <= 11.5.



f. **What are the 5 important variables in building the tree?**

The most important variables in GridSearchCV are AGE, AFFL, BILL, NEIGHBORHOOD_2.0 and NEIGHBORHOOD_13.0

g. **Report if you see any evidence of model overfitting.**

There is no sign of model overfitting as the train and test accuracy have nearly no difference.

3. **What is the significant difference do you see between these two decision trees models (steps 2.1 & 2.2)? How do they compare performance-wise? Explain why those changes may have happened.**

A: As this report mentioned above, the major difference between those decision tree models is classification accuracy. GridSearchCV outperforms default decision tree. Its test accuracy is higher than default tree. Additionally, the default tree is more likely to produce model overfitting and consequently result in low performance on test data.

GridSearchCV tunes the model using hyper parameters to construct a better model. It selects the optimal parameters from a wide range of values to produce an ideal result.

4. **From the better model, can you identify which customers to target for further marketing? Can you provide some descriptive summary of those customers?**

A: The optimal decision tree shows several characteristics of the prospective organic product buyers. The major benefit that people can obtain from decision tree is the quantity of samples. In fact, it shows that a large proportion of customers whose age are over 44 years old and their affluence grade are lower than 12 are likely to purchase the organic products. This means that the customers with this characteristic should be the target group. Therefore, the manager should focus on this target groups for further marketing.

Task 3. Predictive Modelling Using Regression

1. **In preparation for regression, apply transformation method(s) to the variable(s) that need it. List the variables that needed it.**

A: Since we first import a pandas Data Frame object, the .as_matrix() function is used to convert all input variables into a numpy matrix which can be consumed by sklearn. Then, because regression models are sensitive for input variables on different scales, we utilize StandardScaler() method in sklearn to standardise input variables , which ensure all of them are on the same scale. The below pictures show the code for two methods.

```
y = df["ORGYN"]
X = df.drop(["ORGYN"], axis = 1)
rs = 10
X_mat = X.as_matrix()
X_train, X_test, y_train, y_test = train_test_split(X_mat, y, test_size = 0.3, stratify = y, random_state = rs)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
print("Before scaling\n-----------")
for i in range(5):
    col = X_train[:,i]
    print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".
          format(i, min(col), max(col), np.mean(col), np.std(col)))
X_train = scaler.fit_transform(X_train, y_train)
print("After scaling\n-----------")
for i in range(5):
    col = X_train[:,i]
    print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".
          format(i, min(col), max(col), np.mean(col), np.std(col)))
X_test = scaler.transform(X_test)
```

```
Before scaling
-----------
Variable #0: min 18.0, max 79.0, mean 53.83 and std dev 13.21
Variable #1: min 1.5399999999999998, max 239542.13, mean 6347.31 and std dev 6964.55
Variable #2: min 0.0, max 3.0, mean 1.07 and std dev 0.85
Variable #3: min 1.0, max 30.0, mean 8.69 and std dev 3.32
Variable #4: min 0.0, max 39.0, mean 6.57 and std dev 4.62
After scaling
-----------
Variable #0: min -2.7122802775064354, max 1.905749393427998, mean -0.00 and std dev 1.00
Variable #1: min -0.9111525443616153, max 33.48310897793287, mean -0.00 and std dev 1.00
Variable #2: min -1.253180719934095, max 2.2745863460278892, mean 0.00 and std dev 1.00
Variable #3: min -2.316913853961 1036, max 6.415932518111026, mean 0.00 and std dev 1.00
Variable #4: min -1.4240005334369745, max 7.0231277674407995, mean 0.00 and std dev 1.00
```

Furthermore, the following picture lists the variables used in regression model.

```
Data columns (total 80 columns):
AGE                 21483 non-null int32
BILL                21483 non-null float64
CLASS               21483 non-null int64
ORGYN               21483 non-null int64
AFFL                21483 non-null float64
LTIME               21483 non-null float64
TV_REG_Border       21483 non-null uint8
TV_REG_C Scotland   21483 non-null uint8
TV_REG_East         21483 non-null uint8
TV_REG_London       21483 non-null uint8
TV_REG_Midlands     21483 non-null uint8
TV_REG_N East       21483 non-null uint8
TV_REG_N Scot       21483 non-null uint8
TV_REG_N West       21483 non-null uint8
TV_REG_S & S East   21483 non-null uint8
TV_REG_S West       21483 non-null uint8
TV_REG_Ulster       21483 non-null uint8
TV_REG_Wales & West 21483 non-null uint8
TV_REG_Yorkshire    21483 non-null uint8
NEIGHBORHOOD_1.0    21483 non-null uint8
NEIGHBORHOOD_10.0   21483 non-null uint8
NEIGHBORHOOD_11.0   21483 non-null uint8
NEIGHBORHOOD_12.0   21483 non-null uint8
NEIGHBORHOOD_13.0   21483 non-null uint8
NEIGHBORHOOD_14.0   21483 non-null uint8
NEIGHBORHOOD_15.0   21483 non-null uint8
NEIGHBORHOOD_16.0   21483 non-null uint8
NEIGHBORHOOD_17.0   21483 non-null uint8
NEIGHBORHOOD_18.0   21483 non-null uint8
NEIGHBORHOOD_19.0   21483 non-null uint8
NEIGHBORHOOD_2.0    21483 non-null uint8
NEIGHBORHOOD_20.0   21483 non-null uint8
NEIGHBORHOOD_21.0   21483 non-null uint8
NEIGHBORHOOD_22.0   21483 non-null uint8
NEIGHBORHOOD_23.0   21483 non-null uint8
NEIGHBORHOOD_24.0   21483 non-null uint8
NEIGHBORHOOD_25.0   21483 non-null uint8
NEIGHBORHOOD_26.0   21483 non-null uint8
NEIGHBORHOOD_27.0   21483 non-null uint8
NEIGHBORHOOD_28.0   21483 non-null uint8
NEIGHBORHOOD_29.0   21483 non-null uint8
NEIGHBORHOOD_3.0    21483 non-null uint8
NEIGHBORHOOD_30.0   21483 non-null uint8
NEIGHBORHOOD_31.0   21483 non-null uint8
NEIGHBORHOOD_32.0   21483 non-null uint8
NEIGHBORHOOD_33.0   21483 non-null uint8
NEIGHBORHOOD_34.0   21483 non-null uint8
NEIGHBORHOOD_35.0   21483 non-null uint8
NEIGHBORHOOD_36.0   21483 non-null uint8
NEIGHBORHOOD_37.0   21483 non-null uint8
NEIGHBORHOOD_38.0   21483 non-null uint8
NEIGHBORHOOD_39.0   21483 non-null uint8
NEIGHBORHOOD_4.0    21483 non-null uint8
NEIGHBORHOOD_40.0   21483 non-null uint8
NEIGHBORHOOD_41.0   21483 non-null uint8
NEIGHBORHOOD_42.0   21483 non-null uint8
NEIGHBORHOOD_43.0   21483 non-null uint8
NEIGHBORHOOD_44.0   21483 non-null uint8
NEIGHBORHOOD_45.0   21483 non-null uint8
NEIGHBORHOOD_46.0   21483 non-null uint8
NEIGHBORHOOD_47.0   21483 non-null uint8
NEIGHBORHOOD_48.0   21483 non-null uint8
NEIGHBORHOOD_49.0   21483 non-null uint8
NEIGHBORHOOD_5.0    21483 non-null uint8
NEIGHBORHOOD_50.0   21483 non-null uint8
NEIGHBORHOOD_51.0   21483 non-null uint8
NEIGHBORHOOD_52.0   21483 non-null uint8
NEIGHBORHOOD_53.0   21483 non-null uint8
NEIGHBORHOOD_54.0   21483 non-null uint8
NEIGHBORHOOD_55.0   21483 non-null uint8
NEIGHBORHOOD_6.0    21483 non-null uint8
NEIGHBORHOOD_7.0    21483 non-null uint8
NEIGHBORHOOD_8.0    21483 non-null uint8
NEIGHBORHOOD_9.0    21483 non-null uint8
NEIGHBORHOOD_nan    21483 non-null uint8
REGION_Midlands     21483 non-null uint8
REGION_North        21483 non-null uint8
REGION_Scottish     21483 non-null uint8
REGION_South East   21483 non-null uint8
REGION_South West   21483 non-null uint8
```

The final method is logarithmic transformation function. As regression models are easy to be influenced by outlying values in the input variables, we utilize pandas.apply() to apply np.log() method to transform six columns, which including 'AGE', 'BILL', 'CLASS', 'LTIM' , 'AFFL', 'TV_REG_Border', for making these columns normalise distributed. The following image shows the code:

```python
import numpy as np

# list columns to be transformed
columns_to_transform = ['AGE', 'BILL', 'CLASS', 'LTIME',
                        'AFFL','TV_REG_Border']

# copy the dataframe
df_log = df.copy()

# transform the columns with np.log
for col in columns_to_transform:
    df_log[col] = df_log[col].apply(lambda x: x+1)
    df_log[col] = df_log[col].apply(np.log)
```

2. **Build a regression model using the default regression method with all inputs. Once you done it, build another one and tune it using GridSearchCV. Answer the followings:**

    a. **Name the regression function used.**

    A: For this classification task , the logistic regression function is employed to build a model. In sklearn, sklearn.linear_model.LogisticRegression can implement logistic regression.

b. **How much was the difference in performance of two models build, default and optimal?**

A: There are no significant difference between two models. The below two pictures show the performance of default and optimal regression model.

Default model:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=rs)
model.fit(X_train, y_train)
print("Train accuracy:", model.score(X_train, y_train))
print("Test accuracy:", model.score(X_test, y_test))
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Train accuracy: 0.7993749168772443
Test accuracy: 0.8038789759503491
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.71      0.35      0.47      1591

avg / total       0.79      0.80      0.78      6445
```

Optimal model:

```
# grid search CV
params = {'C': [pow(10, x) for x in range(-6, 4)]}
cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_state=rs), cv=10, n_jobs=-1)
cv.fit(X_train_log, y_train_log)
# test the best model
print("Train accuracy:", cv.score(X_train_log, y_train_log))
print("Test accuracy:", cv.score(X_test_log, y_test_log))
y_pred = cv.predict(X_test_log)
print(classification_report(y_test_log, y_pred))
# print parameters of the best model
print(cv.best_params_)
```

```
Train accuracy: 0.8013698630136986
Test accuracy: 0.8054305663304887
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.71      0.36      0.48      1591

avg / total       0.79      0.81      0.78      6445

{'C': 1}
```

From the picture presented above, we can discover the optimal model and default model provide similar test accuracy. Optimal model only increases slightly training and test accuracy.

c. **Show the set parameters for the best model. What are the parameters used? Explain your decision. What are the optimal parameters?**

A: The set parameters are demonstrated as below.

```python
# grid search CV
params = {'C': [pow(10, x) for x in range(-6, 4)]}
cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_state=rs), cv=10, n_jobs=-1)
cv.fit(X_train_log, y_train_log)
```

```
GridSearchCV(cv=10, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=10, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'C': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

We selected hyperparameter C for logistic regression. The hyperparameter C denotes the inverse of regularisation strength, which aids logistic regression to avoid overfitting. According to the image shown above, there are ten hyperparameter C in logistic regression . The GridSearchCV can select the best C in these ten hyperparameter C (1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000). The result show when C = 1, the performance is best. Therefore, the optimal parameter is C = 1.

d. **Report which variables are included in the regression model.**

A: The following picture shows the variables contained in the regression model

```python
feature_names = X.columns
coef = model.coef_[0]
for i in range(len(coef)):
    print(feature_names[i], ':', coef[i])
```

```
AGE : -0.727479769510687
BILL : -0.013584785169338566
CLASS : -0.01418368257286851
AFFL : 0.8358388339810255
LTIME : 0.021499764411505626
TV_REG_Border : -0.04390005482037559
TV_REG_C Scotland : 0.013586632118982972
TV_REG_East : 0.002164228936087834
TV_REG_London : 0.016671509744788465
TV_REG_Midlands : -0.002169663615708612
TV_REG_N East : 0.006564318248928616
TV_REG_N Scot : -0.008342284812697843
TV_REG_N West : 0.002830316723613511
TV_REG_S & S East : -0.009478799162355533
TV_REG_S West : 0.005945317963655542
TV_REG_Ulster : -0.007675261570722381
TV_REG_Wales & West : 0.016048241095778036
TV_REG_Yorkshire : -0.03366183683380679
NEIGHBORHOOD_1.0 : 0.009587177745480554
NEIGHBORHOOD_10.0 : 0.0036261793894213573
NEIGHBORHOOD_11.0 : 0.03624668290054254
NEIGHBORHOOD_12.0 : -0.02927834831663323
NEIGHBORHOOD_13.0 : -0.0025316153762612722
NEIGHBORHOOD_14.0 : -0.0038981803984628628
NEIGHBORHOOD_15.0 : 0.004246723493978367
NEIGHBORHOOD_16.0 : 0.03146116359464345
NEIGHBORHOOD_17.0 : -0.0324144410576749
NEIGHBORHOOD_18.0 : 0.01446209321758129
NEIGHBORHOOD_19.0 : -0.010683769075863665
NEIGHBORHOOD_2.0 : 0.012579753890994878
NEIGHBORHOOD_20.0 : 0.011529318306126005
NEIGHBORHOOD_21.0 : 0.007429524556703627
NEIGHBORHOOD_22.0 : -0.014242720459634999
NEIGHBORHOOD_23.0 : 0.020272943594363584
NEIGHBORHOOD_24.0 : 0.007929625464230068
NEIGHBORHOOD_25.0 : 0.017005576674239554
NEIGHBORHOOD_26.0 : 0.022804925403657635
NEIGHBORHOOD_27.0 : 0.006954646497724075
NEIGHBORHOOD_28.0 : -0.017492982994752970
NEIGHBORHOOD_29.0 : 0.0031349512875080724
NEIGHBORHOOD_3.0 : -0.02445839390899703
NEIGHBORHOOD_30.0 : 0.025124542858756958
NEIGHBORHOOD_31.0 : -0.008355476750332940
NEIGHBORHOOD_32.0 : -0.02221418467380890
NEIGHBORHOOD_33.0 : -0.022809831683985440
NEIGHBORHOOD_34.0 : -0.018470476614138625
NEIGHBORHOOD_35.0 : -0.011347661072298213
NEIGHBORHOOD_36.0 : 0.006555049171498444
NEIGHBORHOOD_37.0 : -0.0012481102565306630
NEIGHBORHOOD_38.0 : -0.0046919127888942890
NEIGHBORHOOD_39.0 : -0.003921877775364348
NEIGHBORHOOD_4.0 : -0.034608685143409854
NEIGHBORHOOD_40.0 : 0.023216796094958120
NEIGHBORHOOD_42.0 : 0.012436229244603051
NEIGHBORHOOD_43.0 : 0.02711526706888311
NEIGHBORHOOD_44.0 : -0.005214329683949786
NEIGHBORHOOD_45.0 : -0.003362816057759825
NEIGHBORHOOD_46.0 : 0.008078099433941926
NEIGHBORHOOD_47.0 : 0.03212027756994168
NEIGHBORHOOD_48.0 : 0.014775366313811544
NEIGHBORHOOD_49.0 : -0.004672086381909114
NEIGHBORHOOD_5.0 : -0.028590957924546058
NEIGHBORHOOD_50.0 : -0.038369693690125050
NEIGHBORHOOD_51.0 : 0.024875575373988164
NEIGHBORHOOD_52.0 : 0.008700396668120655
NEIGHBORHOOD_53.0 : 0.008612229077287259
NEIGHBORHOOD_54.0 : 0.01440908242922054
NEIGHBORHOOD_55.0 : -0.004559165835277261
NEIGHBORHOOD_6.0 : 0.019113880766849268
NEIGHBORHOOD_7.0 : -0.019620844485643333
NEIGHBORHOOD_8.0 : 0.002556870892322813
NEIGHBORHOOD_9.0 : 0.000714304611092643
NEIGHBORHOOD_nan : 0.022090186139113710
REGION_Midlands : 0.007268293245017533
REGION_North : -0.015770506453967083
REGION_Scottish : -0.010806106796020348
REGION_South East : 0.009285388983207011
REGION_South West : 0.005945317963655542
```

e. **Report the top-5 important variables (in the order) in the model.**

A: The top-5 important variables include AFFL, AGE, TV_REG_Border, NEIGHBOOD_50.0, NEIGHBORHOOD_11.0.

```
coef = model.coef_[0]
feature_names = X_log.columns
indices = np.argsort(np.absolute(coef))
indices = np.flip(indices, axis=0)
indices = indices[:5]
for i in indices:
    print(feature_names[i], ':', coef[i])
```

```
AFFL : 0.8358388339810255
AGE : -0.727479769510687
TV_REG_Border : -0.04390005482037559
NEIGHBORHOOD_50.0 : -0.03836969369012505
NEIGHBORHOOD_11.0 : 0.03624668290054254
```

f. **What is classification accuracy on training and test datasets?**

A: Default model:

Train accuracy: 0.7993749168772443

Test accuracy: 0.8038789759503491

Optimal model:

Train accuracy: 0.8013698630136986

Test accuracy: 0.8054305663304887

g. **Report any sign of overfitting.**

A: Since the test accuracy is slightly higher than train accuracy, there is no sign of overfitting in these two models.

3. **Build another regression model using the subset of inputs selected by RFE and selection by model methods. Answer the followings:**

a. **Report which variables are included in the regression model.**

A: For recursive feature elimination, two variables are used containing AGE and AFFL.

```
# running RFE + log transformation
rfe = RFECV(estimator = LogisticRegression(random_state=rs), cv=10)
rfe.fit(X_train_log, y_train_log) # run the RFECV on log transformed dataset

# comparing how many variables before and after
print("Original feature set", X_train_log.shape[1])
print("Number of features after elimination", rfe.n_features_)
```

```
Original feature set 79
Number of features after elimination 2
```

For selection by model method, we use decision tree to select the prominent features for regression model. It shows that AFFL and AGE are two most significant variables.

b. **Report the top-5 important variables (in the order) in the model.**

A: For this regression model, it has only two important variables which are FEEL and AGE. Another top-5 important variables show in below pictures.

```
from data_preprocessing import analyse_feature_importance
analyse_feature_importance(cv.best_estimator_, X_log.columns)
```

```
AGE : 0.5840349886240985
AFFL : 0.40681654234729003
BILL : 0.007398910993758063
REGION_North : 0.0017495580348533574
NEIGHBORHOOD_2.0 : 0.0
NEIGHBORHOOD_13.0 : 0.0
NEIGHBORHOOD_14.0 : 0.0
NEIGHBORHOOD_15.0 : 0.0
NEIGHBORHOOD_16.0 : 0.0
```

c. **What are the parameters used? Explain your decision. What are the optimal parameters? Which regression function is being used?**

A: We use hyper parameter C to help our regression model to avoid overfitting. The optimal C parameters is 1, which show in the below picture.

```
# running RFE + log transformation
rfe = RFECV(estimator = LogisticRegression(random_state=rs), cv=10)
rfe.fit(X_train_log, y_train_log) # run the RFECV on log transformed dataset
# comparing how many variables before and after
print("Original feature set", X_train_log.shape[1])
print("Number of features after elimination", rfe.n_features_)
# select features from log transformed dataset
X_train_sel_log = rfe.transform(X_train_log)
X_test_sel_log = rfe.transform(X_test_log)
# init grid search CV on transformed dataset
params = {'C': [pow(10, x) for x in range(-6, 4)]}
cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_state=rs), cv=10, n_jobs=-1)
cv.fit(X_train_sel_log, y_train_log)
# test the best model
print("Train accuracy:", cv.score(X_train_sel_log, y_train_log))
print("Test accuracy:", cv.score(X_test_sel_log, y_test_log))
y_pred_log = cv.predict(X_test_sel_log)
print(classification_report(y_test_log, y_pred_log))
# print parameters of the best model
print(cv.best_params_)
```

```
Original feature set 79
Number of features after elimination 2
Train accuracy: 0.8012368666046017
Test accuracy: 0.8058960434445307
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.71      0.36      0.48      1591

avg / total       0.79      0.81      0.78      6445

{'C': 1}
```

In recursive feature elimination (RFE) model, we first apply sklearn.feature_selection.RFECV to initiate the RFE with a logistic regression and select the important features. Then, .transform() function is employed to take the selected features into the input set. Finally, Run GridSearchCV function to build the new regression model.

In feature selection using decision tree model, we first initiate a GridSearchCV with DecisionTreeClassifier() function, which can establish decision tree. Afterwards, we utilize `analyse_feature_importance()` function to find the essential features. Next, import SelectFromModel() function from sklearn.feature_selection to choose the important features for training regression model. Finally, build new logistic regression model from above data set containing only two significant features.

d. **Report any sign of overfitting.**

The test accuracy is slightly higher than train accuracy, so there are no overfitting happen in the models.

e. **What is classification accuracy on training and test datasets?**

```
Train accuracy: 0.8012368666046017Test accuracy: 0.8058960434445307
```

4. **Using the comparison statistics, which of the regression models appears to be better? Is there any difference between two models (i.e one with selected variables and another with all variables)? Explain why those changes may have happened.**

A: The regression models with selected variables (RFE and select by other models) appear better, because their train and test accuracy improve slightly compared with models used all variables (default and optimal). However, in three regression models that use hyperparameter C =1, they Test accuracy almost same around 0.805. It means C is a good parameter for increasing accuracy. During running these models, we discover one with selected variables costing less time compared to another with all variables. The reason is that it decreases lots of uncorrelated input variable sets (77 original input sets), which only need 2 input variable sets (AFFL and AGE). Therefore, it save a great deal of computation resources. In the conclusion, there are no significant changes between two models, since all these models just find only AFFL and AGE have high relationship with target variable ORGYN. Thereby, they build similar regression models.

5. **From the better model, can you identify which customers to target? Can you provide some descriptive summary of those customers?**

A: From above regression model, we find AFFL is positive coefficient for ORGYN, which represent the customers who have higher AFFL (Affluence grade on a scale from 1 to 30) level prefer to buy organics food. Moreover, the AGE is negative coefficient for target variable, which means the older customers do not like to buy organic food. To summary, the customers who have high salary that AFFL is bigger than 10 and who is around 35 to 45 years old prefer to buy organics food.

Task 4. Predictive Modelling Using Neural Networks

1. **Build a Neural Network model using the default setting. After that, tune it with GridSearchCV. Answer the following:**

a. **What are the parameters used? Explain your decision. What is the optimal network architecture?**

A: For the parameters, we use “hidden_layer_sizes” and “alpha” which both parameters are hyperparameters.

The “hidden_layer_sizes” is an important parameter in neural network that represents the complexity of the model. It also represents the number of the neurons included in each hidden layer.

The “alpha” is the parameter that is the learning rate of the gradient descent algorithm. It is also used for activation function in each neuron.

In default setting, two parameters have been tested:

```
model=MLPClassifier(random_state=rs)
model.fit(X_train, Y_train)
print("Train Accuracy: ",model.score(X_train,Y_train),"\nTest Accuracy: ",model.score(X_test,Y_test),"\n\n")

Y_pred=model.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(model)
```

```
Train Accuracy:  0.8327570155605799
Test Accuracy:  0.7889837083010085


             precision    recall  f1-score   support

          0       0.83      0.91      0.87      4854
          1       0.61      0.42      0.49      1591

avg / total       0.77      0.79      0.77      6445

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

Which the “alpha” is 0.0001 and “hidden_layer_sizes” is 100.

When setting the max_iter=100, the result is same:

```
model=MLPClassifier(random_state=rs,max_iter=100)
model.fit(X_train, Y_train)
print("Train Accuracy: ",model.score(X_train,Y_train),"\nTest Accuracy: ",model.score(X_test,Y_test),"\n\n")

Y_pred=model.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(model)
```

```
Train Accuracy:  0.8327570155605799
Test Accuracy:  0.7889837083010085


             precision    recall  f1-score   support

          0       0.83      0.91      0.87      4854
          1       0.61      0.42      0.49      1591

avg / total       0.77      0.79      0.77      6445

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=100, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

The solver is using the “adam” algorithm. There is no warning message that means the solver have finished the optimising. That model can be viewed the optimal model.

In GridSearchCV, the value of both parameters is needed to identify by following steps.

First, finding the number of the features in dataset.

```
print(X_train.shape)
```

```
(15038, 82)
```

82 features. Using that result to test the “hidden_layer_sizes” which in the range [1, 82] by using the for loop, and the increment is 10

```
params={'hidden_layer_sizes':[(i,) for i in range(1,82,10)]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train,Y_train)

print("Train Accuracy: ", cv.score(X_train,Y_train))
print("Test Accuracy: ", cv.score(X_test,Y_test))

Y_pred=cv.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(cv.best_params_)
```

```
Train Accuracy:  0.8038302965819922
Test Accuracy:  0.799689681923972
             precision    recall  f1-score   support

          0       0.83      0.93      0.87      4854
          1       0.65      0.40      0.50      1591

avg / total       0.78      0.80      0.78      6445

{'hidden_layer_sizes': (11,)}
```

At the same time, testing the "alpha" by using value [0.1, 0.01, 0.001, 0.0001]

The result:

```
params={'alpha':[0.1,0.01,0.001,0.0001]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train,Y_train)

print("Train Accuracy: ", cv.score(X_train,Y_train))
print("Test Accuracy: ", cv.score(X_test,Y_test))

Y_pred=cv.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(cv.best_params_)
```

```
Train Accuracy:  0.8256417076738928
Test Accuracy:  0.7945694336695113
             precision    recall  f1-score   support

          0       0.82      0.93      0.87      4854
          1       0.64      0.39      0.48      1591

avg / total       0.78      0.79      0.78      6445

{'alpha': 0.1}
```

The 0.1 is the result, that means the value might large than 0.1. Thus, the second test can use large number and test both parameters together.

```
params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,)], 'alpha':[0.5,0.4,0.3,0.2,0.1,0.01]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train,Y_train)

print("Train Accuracy: ", cv.score(X_train,Y_train))
print("Test Accuracy: ", cv.score(X_test,Y_test))

Y_pred=cv.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(cv.best_params_)
```

```
Train Accuracy:  0.8027663253092167
Test Accuracy:  0.8049650892164468
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.69      0.38      0.49      1591

avg / total       0.79      0.80      0.78      6445

{'alpha': 0.3, 'hidden_layer_sizes': (3,)}
```

Ultimately, the result is obtained that the "hidden_layer_sizes" is 3 and the "alpha" is 0.3.

b. **How many iterations are needed to train this network?**

A: The default setting is using 200 maximum iterations and no error message. After testing, the max iterations 75 will report the error message:

```
model=MLPClassifier(random_state=rs,max_iter=75)
model.fit(X_train, Y_train)
print("Train Accuracy: ",model.score(X_train,Y_train),"\nTest Accuracy: ",model.score(X_test,Y_test),"\n\n")

Y_pred=model.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(model)
```

```
Train Accuracy:  0.8342864742651949
Test Accuracy:  0.7896043444530644


             precision    recall  f1-score   support

          0       0.82      0.92      0.87      4854
          1       0.62      0.39      0.48      1591

avg / total       0.77      0.79      0.77      6445

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=75, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

```
C:\Users\n9778977\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:564: Conv
ergenceWarning: Stochastic Optimizer: Maximum iterations (75) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

When setting the value at 76:

```
model=MLPClassifier(random_state=rs,max_iter=76)
model.fit(X_train, Y_train)
print("Train Accuracy: ",model.score(X_train,Y_train),"\nTest Accuracy: ",model.score(X_test,Y_test),"\n\n")

Y_pred=model.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(model)
```

```
Train Accuracy:  0.8327570155605799
Test Accuracy:  0.7889837083010085


             precision    recall  f1-score   support

          0       0.83      0.91      0.87      4854
          1       0.61      0.42      0.49      1591

avg / total       0.77      0.79      0.77      6445

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=76, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

Thus, the network need to train 76 times.

c. **Do you see any sign of over-fitting?**

A: The accuracy of both train data and test are similar which the test size is 30%.

```
model=MLPClassifier(random_state=rs)
model.fit(X_train, Y_train)
print("Train Accuracy: ",model.score(X_train,Y_train),"\nTest Accuracy: ",model.score(X_test,Y_test),"\n\n")
```

```
Train Accuracy:  0.8327570155605799
Test Accuracy:  0.7889837083010085
```

It seems that has little over-fitting but not significant.

The value of “rs” is 10. Actually, this value have less impact on the accuracy. For proving this, we run a loop to get the result below:

```
for i in range(1,10,1):
    model=MLPClassifier(random_state=i)
    model.fit(X_train, Y_train)
    print("Train %d Accuracy: "%(i),model.score(X_train,Y_train),"\nTest %d Accuracy: "%(i),model.score(X_test,Y_test),"\n\n")
```

```
Train 1 Accuracy:  0.8350844527197766
Test 1 Accuracy:  0.791311093871218


Train 2 Accuracy:  0.8306290730150286
Test 2 Accuracy:  0.7945694336695113


Train 3 Accuracy:  0.8377443809017157
Test 3 Accuracy:  0.7860356865787432


Train 4 Accuracy:  0.836547413219843
Test 4 Accuracy:  0.7872769588828549


Train 5 Accuracy:  0.8376778826971671
Test 5 Accuracy:  0.7843289371605896


Train 6 Accuracy:  0.8333554994015162
Test 6 Accuracy:  0.7927075252133436


Train 7 Accuracy:  0.8271711663785077
Test 7 Accuracy:  0.7976726144297905


Train 8 Accuracy:  0.8318260406969011
Test 8 Accuracy:  0.7922420480993018


Train 9 Accuracy:  0.8349514563106796
Test 9 Accuracy:  0.7894491854150504
```

d. **Did the training process converge and resulted in the best model?**

A:

```
model=MLPClassifier(random_state=rs)
model.fit(X_train, Y_train)
print("Train Accuracy: ",model.score(X_train,Y_train),"\nTest Accuracy: ",model.score(X_test,Y_test),"\n\n")

Y_pred=model.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(model)
```

```
Train Accuracy:  0.8327570155605799
Test Accuracy:  0.7889837083010085


             precision    recall  f1-score   support

          0       0.83      0.91      0.87      4854
          1       0.61      0.42      0.49      1591

avg / total       0.77      0.79      0.77      6445

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

Based on above, the training process converge and resulted are in the best model.

e. **What is classification accuracy on training and test datasets?**

A: The Accuracy is shown in following image.

```
Y_pred=model.predict(X_test)

print("Train Accuracy: ",model.score(X_train,Y_train),"\nTest Accuracy: ",model.score(X_test,Y_test),"\n\n")
print(classification_report(Y_test,Y_pred))
```

```
Train Accuracy:  0.8327570155605799
Test Accuracy:  0.7889837083010085


             precision    recall  f1-score   support

          0       0.83      0.91      0.87      4854
          1       0.61      0.42      0.49      1591

avg / total       0.77      0.79      0.77      6445
```

2. **Refine this network by tuning it with GridSearchCV. Report the trained model, same as Task 4.1**

A: Some of the results of the GridSearchCV are shown in 4.1.

The two optimal hyperparemeters are found that the “hidden_layer_sizes”= 3 and the “alpha” = 0.3.

The model is shown as follow.

```
#GridSearchCV

params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,)], 'alpha':[0.5,0.4,0.3,0.2,0.1,0.01]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train,Y_train)

print("Train Accuracy: ", cv.score(X_train,Y_train))
print("Test Accuracy: ", cv.score(X_test,Y_test))

Y_pred=cv.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(cv.best_params_)
print(cv)
```

```
Train Accuracy:  0.8027663253092167
Test Accuracy:  0.8049650892164468
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.69      0.38      0.49      1591

avg / total       0.79      0.80      0.78      6445

{'alpha': 0.3, 'hidden_layer_sizes': (3,)}
GridSearchCV(cv=10, error_score='raise',
       estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
```

The log transformation is used to improve the model:

```
params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,)], 'alpha':[0.6,0.5,0.4,0.3,0.2,0.1]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train_log,Y_train_log)

print("Train Accuracy: ", cv.score(X_train_log,Y_train_log))
print("Test Accuracy: ", cv.score(X_test_log,Y_test_log))

Y_pred_log=cv.predict(X_test_log)
print(classification_report(Y_test_log,Y_pred_log))

print(cv.best_params_)
print(cv)
```

```
Train Accuracy:  0.801702354036441
Test Accuracy:  0.8015515903801397
             precision    recall  f1-score   support

          0       0.82      0.94      0.88      4854
          1       0.68      0.37      0.48      1591

avg / total       0.79      0.80      0.78      6445

{'alpha': 0.4, 'hidden_layer_sizes': (3,)}
GridSearchCV(cv=10, error_score='raise',
       estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'hidden_layer_sizes': [(1,), (2,), (3,), (4,), (5,)], 'alpha': [0.6, 0.5, 0.4, 0.3, 0.2, 0.1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

The accuracy is similar before transforming by log. Now, this architecture can be viewed the optimal model.

The iteration:

```
params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,)], 'alpha':[0.6,0.5,0.4,0.3,0.2,0.1]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs,max_iter=70),cv=10,n_jobs=-1)
cv.fit(X_train_log,Y_train_log)

print("Train Accuracy: ", cv.score(X_train_log,Y_train_log))
print("Test Accuracy: ", cv.score(X_test_log,Y_test_log))

Y_pred_log=cv.predict(X_test_log)
print(classification_report(Y_test_log,Y_pred_log))

print(cv.best_params_)
```

```
Train Accuracy:  0.801702354036441
Test Accuracy:  0.8015515903801397
             precision    recall  f1-score   support

          0       0.82      0.94      0.88      4854
          1       0.68      0.37      0.48      1591

avg / total       0.79      0.80      0.78      6445

{'alpha': 0.4, 'hidden_layer_sizes': (3,)}
```

The network can be trained 70 times.

The accuracy as mentioned above, there is no significant overfitting shown by the model based on both accuracies are around 80%.

```
#GridSearchCV

params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,)], 'alpha':[0.5,0.4,0.3,0.2,0.1,0.01]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train,Y_train)

print("Train Accuracy: ", cv.score(X_train,Y_train))
print("Test Accuracy: ", cv.score(X_test,Y_test))

Y_pred=cv.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(cv.best_params_)
print(cv)
```

```
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.69      0.38      0.49      1591

avg / total       0.79      0.80      0.78      6445

{'alpha': 0.3, 'hidden_layer_sizes': (3,)}
GridSearchCV(cv=10, error_score='raise',
       estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'hidden_layer_sizes': [(1,), (2,), (3,), (4,)], 'alpha': [0.5, 0.4, 0.3, 0.2, 0.1, 0.01]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

The train process converges and results in the best model.

The accuracy before the transformation:

```
print("Train Accuracy: ", cv.score(X_train,Y_train))
print("Test Accuracy: ", cv.score(X_test,Y_test))

Y_pred=cv.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(cv.best_params_)
print(cv)
```

```
Train Accuracy:  0.8027663253092167
Test Accuracy:  0.8049650892164468
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.69      0.38      0.49      1591

avg / total       0.79      0.80      0.78      6445
```

After transformation:

```
print("Train Accuracy: ", cv.score(X_train_log,Y_train_log))
print("Test Accuracy: ", cv.score(X_test_log,Y_test_log))

Y_pred_log=cv.predict(X_test_log)
print(classification_report(Y_test_log,Y_pred_log))

print(cv.best_params_)
```

```
Train Accuracy:  0.801702354036441
Test Accuracy:  0.8015515903801397
             precision    recall  f1-score   support

          0       0.82      0.94      0.88      4854
          1       0.68      0.37      0.48      1591

avg / total       0.79      0.80      0.78      6445
```

3. **Build another Neural Network model with inputs selected from RFE with regression (use the best model generated in Task 3) and selection with decision tree (use the best model from Task 2). Answer the following:**

   a. **Did feature selection help here? Any change in the network architecture? What inputs are being used as the network input?**

   A: Yes, the feature selection increases the model performance.

```
# Recursive Feature Elimination
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression

rfe= RFECV(estimator=LogisticRegression(random_state=rs),cv=10)
rfe.fit(X_train_log,Y_train_log)

print(rfe.n_features_)
```

```
2
```

```
X_train_rfe=rfe.transform(X_train_log)
X_test_rfe=rfe.transform(X_test_log)

params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,)], 'alpha':[0.01,0.001,0.0001,0.00001]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train_rfe,Y_train_log)

print("Train Accuracy: ", cv.score(X_train_rfe,Y_train_log))
print("Test Accuracy: ", cv.score(X_test_rfe,Y_test_log))

Y_pred_log=cv.predict(X_test_rfe)
print(classification_report(Y_test,Y_pred_log))

print(cv.best_params_)
print(cv)
```

```
Train Accuracy:  0.8001063971272776
Test Accuracy:  0.804344453064391
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.70      0.36      0.48      1591

avg / total       0.79      0.80      0.78      6445

{'alpha': 0.0001, 'hidden_layer_sizes': (4,)}
```

The alpha is 0.0001 and the hidden layer size is 4. That is significant change that the previous alpha is 0.3 and the size of hidden layer is 3.

Two features are selected:

```
from sklearn.feature_selection import SelectFromModel

selectmodel = SelectFromModel(cv.best_estimator_,prefit=True)
X_train_sel_model= selectmodel.transform(X_train)
X_test_sel_model= selectmodel.transform(X_test)

print(X_train_sel_model.shape)
```

```
(15038, 2)
```

The network input is using "AGE" and "AFFL" which are related high importance.

```
#Importance
from data_preprocessing import analyse_feature_importance

analyse_feature_importance(cv.best_estimator_, X_log.columns)
```

```
AGE : 0.6033301906318194
AFFL : 0.3830061253252705
BILL : 0.010577496968829325
LTIME : 0.0016354304673546472
CLASS_Tin : 0.0014507566067261518
REGION_South East : 0.0
NEIGHBORHOOD_19.0 : 0.0
NEIGHBORHOOD_12.0 : 0.0
NEIGHBORHOOD_13.0 : 0.0
NEIGHBORHOOD_14.0 : 0.0
NEIGHBORHOOD_15.0 : 0.0
NEIGHBORHOOD_16.0 : 0.0
NEIGHBORHOOD_17.0 : 0.0
NEIGHBORHOOD_18.0 : 0.0
NEIGHBORHOOD_2.0 : 0.0
NEIGHBORHOOD_28.0 : 0.0
NEIGHBORHOOD_20.0 : 0.0
NEIGHBORHOOD_21.0 : 0.0
NEIGHBORHOOD_22.0 : 0.0
NEIGHBORHOOD_23.0 : 0.0
```

b. **What is classification accuracy on training and test datasets? Is there any improvement in the outcome?**

A: The accuracy is similar to the previous result that has no significant change.

```
params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,)], 'alpha':[0.1,0.01,0.001,0.0001,0.00001]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train_sel_model,Y_train)

print("Train Accuracy: ", cv.score(X_train_sel_model,Y_train))
print("Test Accuracy: ", cv.score(X_test_sel_model,Y_test))

Y_pred2=cv.predict(X_test_sel_model)
print(classification_report(Y_test,Y_pred2))

print(cv.best_params_)
```

```
Train Accuracy:  0.8038967947865407
Test Accuracy:  0.8066718386346005
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.70      0.38      0.49      1591

avg / total       0.79      0.81      0.79      6445

{'alpha': 0.001, 'hidden_layer_sizes': (4,)}
```

c. **How many iterations are now needed to train this network?**

A: It is enough to train model with 25 times.

```
params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,)], 'alpha':[0.1,0.01,0.001,0.0001,0.00001]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs,max_iter=25),cv=10,n_jobs=-1)
cv.fit(X_train_sel_model,Y_train)

print("Train Accuracy: ", cv.score(X_train_sel_model,Y_train))
print("Test Accuracy: ", cv.score(X_test_sel_model,Y_test))

Y_pred2=cv.predict(X_test_sel_model)
print(classification_report(Y_test,Y_pred2))

print(cv.best_params_)
```

```
Train Accuracy:  0.8015028594227955
Test Accuracy:  0.8063615205585726
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.70      0.37      0.49      1591

avg / total       0.79      0.81      0.78      6445

{'alpha': 0.001, 'hidden_layer_sizes': (3,)}
```

d. **Do you see any sign of over-fitting?**

A: There is no indication of overfitting. Both accuracy are still around 80%.

e. **Did the training process converge and resulted in the best model?**

A: Yes. The training process is converged for each foregoing process. Therefore, it is considered as the best model.

```
X_train_rfe=rfe.transform(X_train_log)
X_test_rfe=rfe.transform(X_test_log)

params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,)], 'alpha':[0.01,0.001,0.0001,0.00001]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train_rfe,Y_train_log)

print("Train Accuracy: ", cv.score(X_train_rfe,Y_train_log))
print("Test Accuracy: ", cv.score(X_test_rfe,Y_test_log))

Y_pred_log=cv.predict(X_test_rfe)
print(classification_report(Y_test,Y_pred_log))

print(cv.best_params_)
print(cv)
```

```
Train Accuracy:  0.8001063971272776
Test Accuracy:  0.804344453064391
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.70      0.36      0.48      1591

avg / total       0.79      0.80      0.78      6445

{'alpha': 0.0001, 'hidden_layer_sizes': (4,)}
GridSearchCV(cv=10, error_score='raise',
       estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'hidden_layer_sizes': [(1,), (2,), (3,), (4,), (5,)], 'alpha': [0.01, 0.001, 0.0001, 1e-05]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

f. **Finally, see whether the change in network architecture can further improve the performance, use GridSearchCV to tune the network. Report if there was any improvement.**

A:

Part 1: The default setting for model:

```
model=MLPClassifier(random_state=rs)
model.fit(X_train, Y_train)
print("Train Accuracy: ",model.score(X_train,Y_train),"\nTest Accuracy: ",model.score(X_test,Y_test),"\n\n")

Y_pred=model.predict(X_test)
print(classification_report(Y_test,Y_pred))

print(model)
```

```
Train Accuracy:  0.8327570155605799
Test Accuracy:  0.7889837083010085


             precision    recall  f1-score   support

          0       0.83      0.91      0.87      4854
          1       0.61      0.42      0.49      1591

avg / total       0.77      0.79      0.77      6445

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

Part 2: The GridSearchCV for model:

```
params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,)], 'alpha':[0.6,0.5,0.4,0.3,0.2,0.1]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train_log,Y_train_log)

print("Train Accuracy: ", cv.score(X_train_log,Y_train_log))
print("Test Accuracy: ", cv.score(X_test_log,Y_test_log))

Y_pred_log=cv.predict(X_test_log)
print(classification_report(Y_test_log,Y_pred_log))

print(cv.best_params_)
print(cv)
```

```
Train Accuracy:  0.801702354036441
Test Accuracy:  0.8015515903801397
             precision    recall  f1-score   support

          0       0.82      0.94      0.88      4854
          1       0.68      0.37      0.48      1591

avg / total       0.79      0.80      0.78      6445

{'alpha': 0.4, 'hidden_layer_sizes': (3,)}
GridSearchCV(cv=10, error_score='raise',
       estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'hidden_layer_sizes': [(1,), (2,), (3,), (4,), (5,)], 'alpha': [0.6, 0.5, 0.4, 0.3, 0.2, 0.1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

Part 3: The selection input for model:

```
params={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,)], 'alpha':[0.1,0.01,0.001,0.0001,0.00001]}

cv=GridSearchCV(param_grid=params,estimator=MLPClassifier(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train_sel_model,Y_train)

print("Train Accuracy: ", cv.score(X_train_sel_model,Y_train))
print("Test Accuracy: ", cv.score(X_test_sel_model,Y_test))

Y_pred2=cv.predict(X_test_sel_model)
print(classification_report(Y_test,Y_pred2))

print(cv.best_params_)
print(cv)
```

```
Train Accuracy:  0.8038967947865407
Test Accuracy:  0.8066718386346005
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.70      0.38      0.49      1591

avg / total       0.79      0.81      0.79      6445

{'alpha': 0.001, 'hidden_layer_sizes': (4,)}
GridSearchCV(cv=10, error_score='raise',
       estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'hidden_layer_sizes': [(1,), (2,), (3,), (4,), (5,)], 'alpha': [0.1, 0.01, 0.001, 0.0001, 1e-05]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

The changes of part 2 and part 3 are the two parameters. The optimal alpha is from 0.4 reduced to the 0.001. The optimal hidden layer is from 3 increased to 4.

The accuracy has similar results from part 2 and part3. However, there is a big different compared with part 1. This result shows the over-fitting is improved.

The other part in architecture has no significant change.

3. Using the comparison methods, which of the models (i.e one with selected variables and another with all variables) appears to be better?

From the better model, can you identify which customers to target? Can you provide some descriptive summary of those customers? Is it easy to comprehend the performance of the best neural network model for decision making?

A: The results of three models are shown as below:

Decision Tree:

```
#Model comparing

#CV for Decision Tree
params={'criterion':['gini','entropy'],
        'max_depth': range(2,7),
        'min_samples_leaf': range(10,500,10)}

cv=GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=rs),cv=10)
cv.fit(X_train,Y_train)

dt_model=cv.best_estimator_
print(dt_model)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=80, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=10,
            splitter='best')
```

Logistic Regression:

```
#CV for Logistic Regression
params_log_reg={'C':[pow(10,x) for x in range(-6,4)]}
cv=GridSearchCV(param_grid=params_log_reg,estimator=LogisticRegression(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train,Y_train)

log_reg_model= cv.best_estimator_

print(log_reg_model)
```

```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=10, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Neural Network:

```
#CV for Neural Network
params_nn={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,),(6,)], 'alpha':[0.5,0.4,0.3,0.2,0.1]}

cv=GridSearchCV(param_grid=params_nn,estimator=MLPClassifier(max_iter=500,random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train,Y_train)

nn_model=cv.best_estimator_
print(nn_model)
```

```
MLPClassifier(activation='relu', alpha=0.3, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(3,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=500, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=10, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

The accuracy of the models:

```
#Accuracy Testing

#Decision Tree
Y_pred_dt=dt_model.predict(X_test)
Y_pred_log_reg=log_reg_model.predict(X_test)
Y_pred_nn=nn_model.predict(X_test)

print("Accuracy score on test for DT:", accuracy_score(Y_test, Y_pred_dt))
print("Accuracy score on test for logistic regression:", accuracy_score(Y_test, Y_pred_log_reg))
print("Accuracy score on test for NN:", accuracy_score(Y_test, Y_pred_nn))
```

```
Accuracy score on test for DT: 0.8094647013188518
Accuracy score on test for logistic regression: 0.8037238169123352
Accuracy score on test for NN: 0.8049650892164468
```

The accuracy shows the result that decision tress model is better.

```
#Typical
Y_pred=dt_model.predict(X_test)

#From Decision Tree
Y_pred_proba_dt=dt_model.predict_proba(X_test)

print("Probability produced by decision tree for each class vs actual prediction on ORGYN (0 = won't purchase, 1 = will purchase)
print("(Probs on zero)\t\t(probs on one)\t\t(prediction made)")

#Top 10
for i in range(20):
    print(Y_pred_proba_dt[i][0],'\t\t',Y_pred_proba_dt[i][1],'\t\t',Y_pred[i])
```

```
Probability produced by decision tree for each class vs actual prediction on ORGYN (0 = won't purchase, 1 = will purchase). You
should be able to see the default threshold of 0.5.
(Probs on zero)         (probs on one)          (prediction made)
0.5737051792828686               0.4262948207171315              0
0.8513349514563107               0.1486650485436893              0
0.792608695652174                0.2073913043478261              0
0.792608695652174                0.2073913043478261              0
0.792608695652174                0.2073913043478261              0
0.792608695652174                0.2073913043478261              0
0.32589285714285715              0.6741071428571429              1
0.9164345403899722               0.08356545961002786             0
0.32589285714285715              0.6741071428571429              1
0.8513349514563107               0.1486650485436893              0
0.792608695652174                0.2073913043478261              0
0.94875          0.05125                 0
0.7784090909090909               0.2215909090909091              0
0.9164345403899722               0.08356545961002786             0
0.5737051792828686               0.4262948207171315              0
0.792608695652174                0.2073913043478261              0
0.15730337078651685              0.8426966292134831              1
0.6415094339622641               0.3584905660377358              0
0.9164345403899722               0.08356545961002786             0
0.792608695652174                0.2073913043478261              0
```

From the results listed above, the value over 0.5 can be viewed as positive. There are three results showing that the customers should target which depended on the AGE, AFFL and BILL (in order to the importance).

To sum up, based on the two selected features in analysis, the customers who will purchase the product depended on the AGE and AFFL, which related to the wealth. The importance's test prove that those two factors have higher relationship with "ORGYN" than other elements.

```
#Importance
from data_preprocessing import analyse_feature_importance

analyse_feature_importance(cv.best_estimator_, X_log.columns)
```

```
AGE : 0.6033301906318194
AFFL : 0.3830061253252705
BILL : 0.010577496968829325
LTIME : 0.0016354304673546472
CLASS_Tin : 0.0014507566067261518
REGION_South East : 0.0
```

Overall, although those results are similar, the decision tree model can be easily identified as the best model. For the neural network, the model is built but not visualising to the figure. Thus, the decision is not easy to make depended on the pure structure of the neural network. However, there is no doubt that the figure of neural network will be useful for decision making based on clear relationship shown among the input layer, hidden layer and output layer if the figure is generated.

Task 5. Generating an Ensemble Model and Comparing Models

1. **Generate an ensemble model to include the best regression model, best decision tree model, and best neural network model.**

    a. **Does the Ensemble model outperform the underlying models? Resonate your answer.**

    A: In ensemble model, the soft voting is using based on "sklearn"

    The setting is shown as follows:

```
# Ensemble model
from sklearn.ensemble import VotingClassifier

# Initialise the classifier with 3 different estimators
voting = VotingClassifier(estimators=[('dt', dt_model), ('lr', log_reg_model), ('nn', nn_model)], voting='soft')
```

    The result of the ensemble model:

```
# Fit the voting classifier to training data
voting.fit(X_train, Y_train)

# Evaluate train and test accuracy
print("Ensemble train accuracy:", voting.score(X_train, Y_train))
print("Ensemble test accuracy:", voting.score(X_test, Y_test))

# Evaluate ROC auc score
Y_pred_proba_ensemble = voting.predict_proba(X_test)
roc_index_ensemble = roc_auc_score(Y_test, Y_pred_proba_ensemble[:, 1])
print("ROC score of voting classifier:", roc_index_ensemble)
```

```
Ensemble train accuracy: 0.8042292858092831
Ensemble test accuracy: 0.804499612102405
ROC score of voting classifier: 0.7936771969025398
```

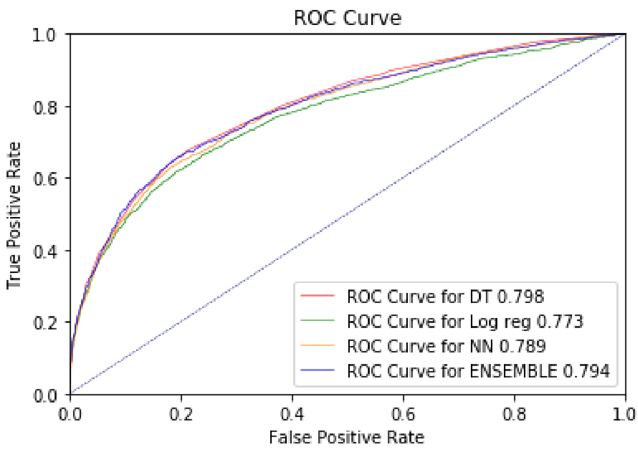
When comparing with other three models:

```
#Accuracy Testing

#Decision Tree
Y_pred_dt=dt_model.predict(X_test)
Y_pred_log_reg=log_reg_model.predict(X_test)
Y_pred_nn=nn_model.predict(X_test)

print("Accuracy score on test for DT:", accuracy_score(Y_test, Y_pred_dt))
print("Accuracy score on test for logistic regression:", accuracy_score(Y_test, Y_pred_log_reg))
print("Accuracy score on test for NN:", accuracy_score(Y_test, Y_pred_nn))
```

```
Accuracy score on test for DT: 0.8094647013188518
Accuracy score on test for logistic regression: 0.8037238169123352
Accuracy score on test for NN: 0.8049650892164468
```

It is obvious that the accuracy is similar to others, but the decision tree is still a better model.

For ROC curve:



Although the curve of ensemble model has a part better than the decision tree, the overall performance of decision tree is better than the ensemble model.

2. **Use the comparison methods to compare the best decision tree model, the best regression model, the best neural network model and the ensemble model.**

   a. **Discuss the findings led by (a) ROC Chart and Index; (b) Accuracy Score; (c) Classification Report.**

A: **a) ROC:**

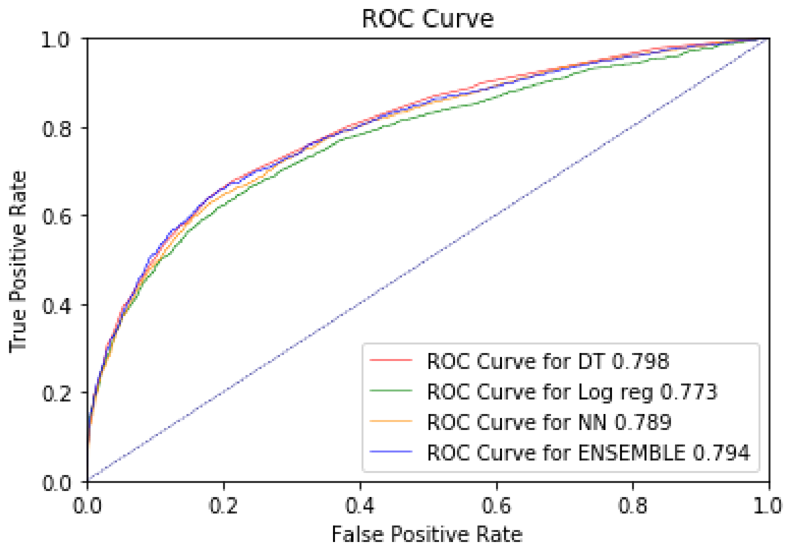
For the ROC index result:

```
from sklearn.metrics import roc_auc_score

Y_pred_proba_dt = dt_model.predict_proba(X_test)
Y_pred_proba_log_reg = log_reg_model.predict_proba(X_test)
Y_pred_proba_nn = nn_model.predict_proba(X_test)

roc_index_dt = roc_auc_score(Y_test, Y_pred_proba_dt[:, 1])
roc_index_log_reg = roc_auc_score(Y_test, Y_pred_proba_log_reg[:, 1])
roc_index_nn = roc_auc_score(Y_test, Y_pred_proba_nn[:, 1])

print("ROC index on test for DT:", roc_index_dt)
print("ROC index on test for logistic regression:", roc_index_log_reg)
print("ROC index on test for NN:", roc_index_nn)
```

```
ROC index on test for DT: 0.7983153331846808
ROC index on test for logistic regression: 0.7729629764872815
ROC index on test for NN: 0.7888111614647391
```

The above has shown the result of the resemble model. The ROC index of the resemble model is 0.794. Thus, the index indicates the decision tree model is better again.

For the ROC graphic:



The ROC curve is also showing the decision tree outperforms others.

b) Accuracy score:

```
#Accuracy Testing

#Decision Tree
Y_pred_dt=dt_model.predict(X_test)
Y_pred_log_reg=log_reg_model.predict(X_test)
Y_pred_nn=nn_model.predict(X_test)

print("Accuracy score on test for DT:", accuracy_score(Y_test, Y_pred_dt))
print("Accuracy score on test for logistic regression:", accuracy_score(Y_test, Y_pred_log_reg))
print("Accuracy score on test for NN:", accuracy_score(Y_test, Y_pred_nn))
```

```
Accuracy score on test for DT: 0.8094647013188518
Accuracy score on test for logistic regression: 0.8037238169123352
Accuracy score on test for NN: 0.8049650892164468
```

```
voting.fit(X_train, Y_train)

# Evaluate train and test accuracy
print("Ensemble train accuracy:", voting.score(X_train, Y_train))
print("Ensemble test accuracy:", voting.score(X_test, Y_test))

# Evaluate ROC auc score
Y_pred_proba_ensemble = voting.predict_proba(X_test)
roc_index_ensemble = roc_auc_score(Y_test, Y_pred_proba_ensemble[:, 1])
print("ROC score of voting classifier:", roc_index_ensemble)
```

```
Ensemble train accuracy: 0.8042292858092831
Ensemble test accuracy: 0.804499612102405
ROC score of voting classifier: 0.7936771969025398
```

It shows the best accuracy among these models is the decision tree.

c) Classification Report:

Decision Tree:

```
#Model comparing

#CV for Decision Tree
params={'criterion':['gini','entropy'],
        'max_depth': range(2,7),
        'min_samples_leaf': range(10,500,10)}

cv=GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=rs),cv=10)
cv.fit(X_train,Y_train)

Y_pred_dt=dt_model.predict(X_test)
print(classification_report(Y_test,Y_pred_dt))

dt_model=cv.best_estimator_
print(dt_model)
```

```
             precision    recall  f1-score   support

          0       0.83      0.95      0.88      4854
          1       0.71      0.39      0.50      1591

avg / total       0.80      0.81      0.79      6445
```

Logistic Regression:

```
#CV for Logistic Regression
params_log_reg={'C':[pow(10,x) for x in range(-6,4)]}
cv=GridSearchCV(param_grid=params_log_reg,estimator=LogisticRegression(random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train,Y_train)

Y_pred_log_reg=log_reg_model.predict(X_test)
print(classification_report(Y_test,Y_pred_log_reg))

log_reg_model= cv.best_estimator_
print(log_reg_model)
```

```
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.71      0.35      0.47      1591

avg / total       0.79      0.80      0.78      6445
```

Neural Network:

```
#CV for Neural Network
params_nn={'hidden_layer_sizes':[(1,),(2,),(3,),(4,),(5,),(6,)], 'alpha':[0.5,0.4,0.3,0.2,0.1]}

cv=GridSearchCV(param_grid=params_nn,estimator=MLPClassifier(max_iter=500,random_state=rs),cv=10,n_jobs=-1)
cv.fit(X_train,Y_train)

Y_pred_nn=nn_model.predict(X_test)
print(classification_report(Y_test,Y_pred_nn))

nn_model=cv.best_estimator_
print(nn_model)
```

```
             precision    recall  f1-score   support

          0       0.82      0.95      0.88      4854
          1       0.69      0.38      0.49      1591

avg / total       0.79      0.80      0.78      6445
```

The classification reports show the similar result. However, the average value of the precision is higher than other models. Again, the result also indicates the decision tree is better.

b. **Do all the models agree on the customers characteristics? How do they vary?**

Yes, all models agree on those customers characteristics which are the AGE and AFFL.

In decision tree model and logistic regression model, the results show the AGE has negative correlation with the ORGYN, which is the target value, and AFFL has positive correlation with the ORGYN where the AFFL is related to the wealth. Those results predict that when the age of the customers is increased that the possibility of purchasing the product will be decreased. For the AFFL, the possibility of the product purchasing is gain when the index of AFFL is raised, and the potential explanation of that might be meaning the high AFFL level representing those customers are able to process the purchase easily.

In neural network, based on the figure of network has not generated that the result is unavailable. However, following the analysis and processing the improvement of the neural network, there are two features obtained after model improved and importing the importance list that the result is showing the AGE, AFFL and BILL are top three elements. Based on the two features, ultimately, the model is predicting the AGE and AFFL are two important customer's characteristics. However, it is difficult to analyse the tendencies of two elements because of no figure generated.

Overall, the decision tree model and logistic regression model show there are two characteristics related to the ORGYN and provide the tendencies. However, the neural network can only predict AGE and AFFL are two important factors.

Task 6. Final Remarks: Decision Making

1. **Finally, based on all models and analysis, is there a particular model you will use in decision making? Justify your choice.**
   A: At end of the report, our group determine to use optimal decision tree model for decision making. The first reason this is a classification problem, which is suitable for decision tree. For instance, there are many variables especially target variable ORGYN that are not continuous variables. The second reason is that the optimal decision tree model have the best accuracy performance according to the task 5 ROC curve. It will give the best predictive results for organics food buyers. The final reason decision tree model is easy to understand and implement. For example, it can handle mixed measurement scales and missing values. Also, it is simple to visualize the whole tree that can help us to make decision for target customers.

2. **Can you summarise positives and negatives of each predictive modelling method based on this analysis?**
   A: For decision tree, the first positives is easy to select two important features for all input variables, which save lots of computation resources and provide reasonable training time. The second advantage is that this model is able to handle large number of features. For instance, there are many different type of information in the data set, like address (NEBOOR…), age and wealth level (AFFL). The third benefit is that the result can be visualized to help us see the relationship between features and target value ORGYN. However, there are two disadvantages for decision tree model. Visualizing trees may be tedious is the one negative. When we use default decision tree to visualize the data, the tree is too big to recognize the useful information. Another shortcoming is to have high instability. It give several decision tree model when the different split variable is applied. Sometimes, the result will become different even use the same dataset.

   For regression models, the first positive is show clear relationship between ORGYN and other input variables via correlation coefficient. Thereby, it have good interpretability and simplicity for us to understand how the relationship change precisely. For example, in our analysis, we find the AFFL is positive coefficient for ORGYN, which means the bigger AFFL level are predicted to be organics food buyers. Conversely, the AGE is negative coefficient for ORGYN, which means the older customers predicted it is less likely to be organics food buyer. Also, other smaller coefficient show that there are no significant influence to our target variable. The second benefit it is a fast application. For the logistic regression model, we just use one hyper parameter C to generate optimal model. It do not need too much programming and running time. However, the regression model cannot handle larger number of missing values, so we delate the gender columns because it have too many missing values. Another drawback, it hard to visualize the logistic regression model, which cannot give us a direct impression for correlation between different variables.

For neural network model, the positive point in this analysis is that can process large size of the data, for example, 82 features in this report, and can be able to test data whether the some values in data are missing or not. However, the drawbacks are also obvious. The neural network take a lot of time to operating the program, especially in testing the two hyper parameters to be optimal. Additionally, in this report, the figure of the neural network has not generated that leading the difficult analysis and it is hard to interpret the structure of neural network for predicting the target state, will buy or not. The hidden layer size is also a problem that large size can bring about the overfitting although it is not significant. Moreover, visualising the neural network requires a good programming skill and an algorithm and that is a key reason for that report cannot analyse more detail from the neural network model.

3. **How the outcome of this study can be used by decision makers?**

A: this outcome show that there are two features are critical for decision makers. On the one hand, the first is AFFL (Affluence grade on a scale from 1 to 30) level, which the customer have higher AFFL level like to eat organics food. Therefore, the decision makers should concentrate on prosperous customers and give convenience for them to buy organics food. On the other hand, the result show the younger customers more like to buy organics food. Hence, the company may advertise their organics food for young customers, such as the working people who is around 35 years old. At last, the outcome demonstrate the region factors do not impact customers to buy organics food, so the organics company do not need to analyse which area's people like to buy organics food. Also, the bill features have no correlation with organics food buyers, which means we cannot predict organics food buyers according to how much the customer's spending money in supermarket. In conclusion, this study discover many feature information do not have relationship for buying organics food. Only, the AGE and AFFL impact to customers buy the organics food. Therefore, the decision makers should focus on these two features to find the potential organics food buyers.